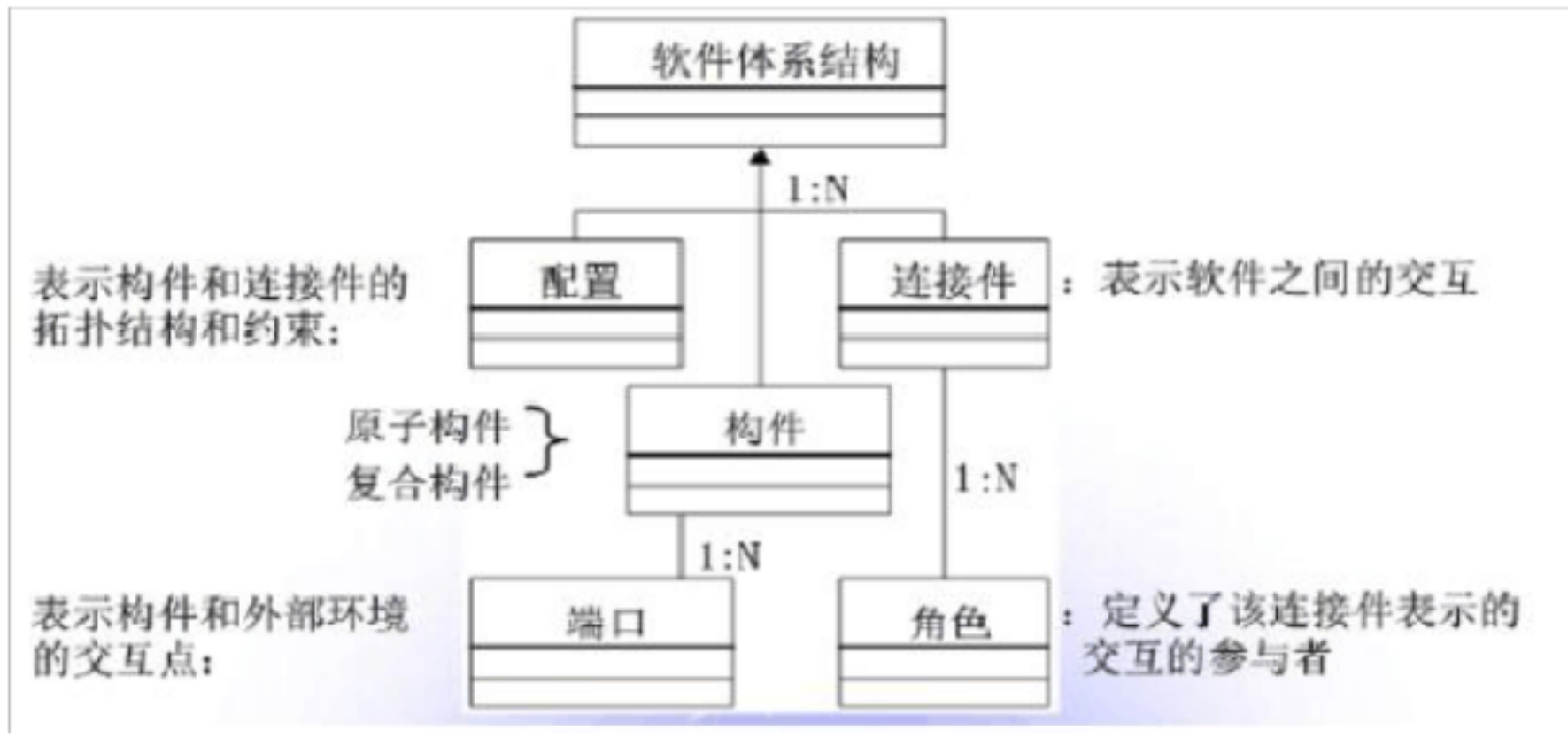
目录

1、 构 件和连接件

软件体系结构的核心模型由五种元素组成： 构件、 连接件、 配置、 端口、 角色。其中， 构件、连接件和配置是最基本的元素。



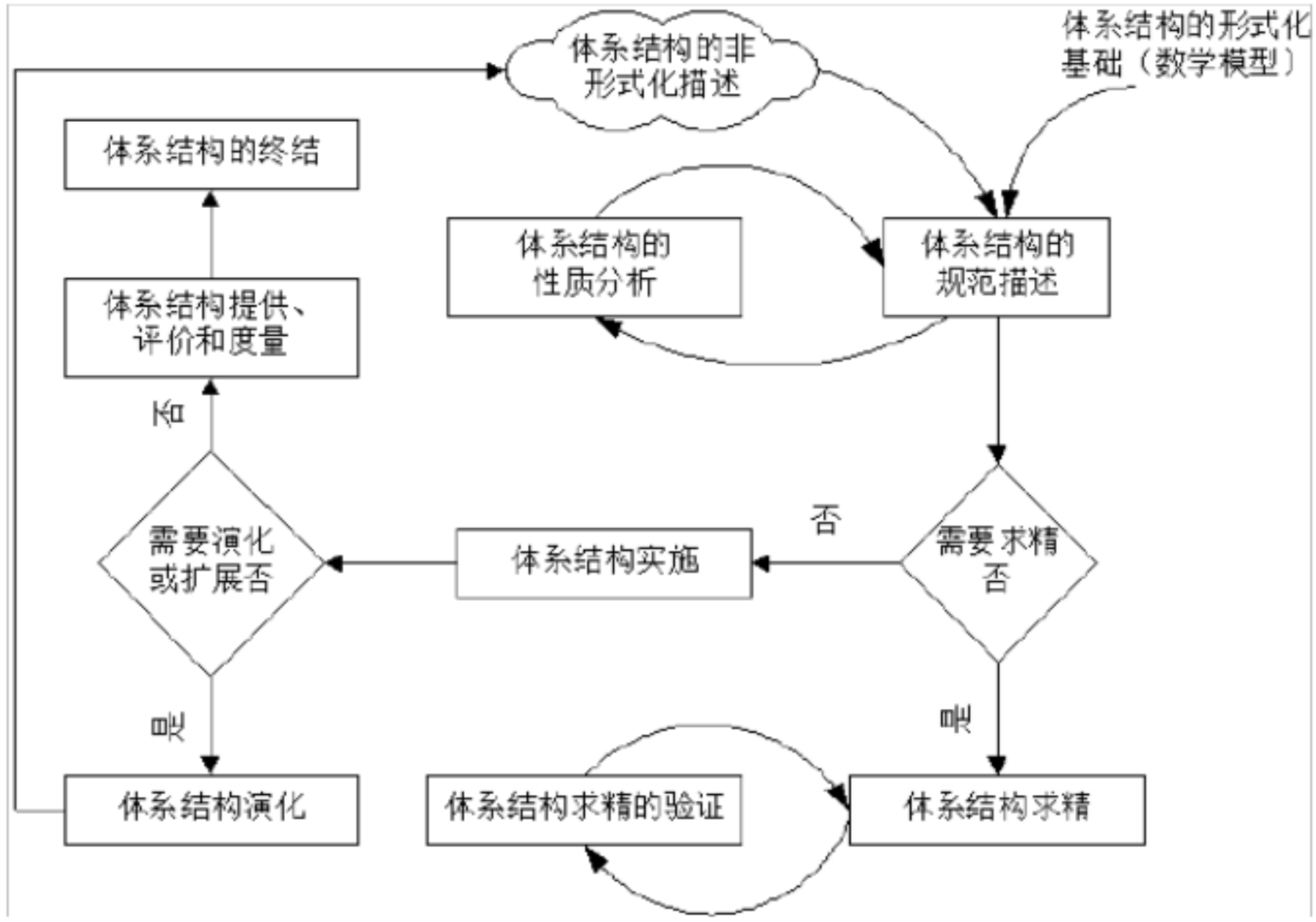
构件： 具有某种功能的可重用软件单元， 表示系统中主要的计算和数据存储。 构件只能通过接口与外部交互， 接口由一组端口组成， 每个端口表示了构件与外部环境的交互点。 通过不同的端口类型，一个构件可以提供多重接口。

（每个构件都有一组输入和输出， 构件读输入的数据流， 经过内部处理， 然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成， 所以在输入被完全消费之前，输出便产生了。 ）

构件的定义： 构件是指语义完整、 语法正确和有可重用价值的单位软件， 是软件重用过程中可以明确辨识的系统；结构上，它是语义描述、通讯接口和实现代码的复合体。

连接件： 表示了构件间的交互； 连接件也有接口， 由一组角色组成，每个角色定义了该连接件表示的交互的参与者。

2、软 件体系结构生命周期模型



模型解释：

1、软件体系结构的非形式化描述　　；

2、软件体系结构规范描述和分析　　；

3、软件体系结构的求精及其验证　　；

4、软件体系结构的实施　　；

5、软件体系结构的演化和扩展　　；

6、软件体系结构提供、评价和度量　　；

7、软件体系结构的终结

3、软件重用技术在软件开发中的作用？

软件重用可以分为：代码重用，设计结果重用，分析结果的重用，测试信息的重用，体系结构的重用。

提高软件生产率，降低软件开发的成本；　　提高软件质量；　互操作性好；

支持原型开发；

提高了软件的可维护性、可扩展性、可理解性。　　举例：1976　年　IBM 、HP 、NEC、 AT&T

的调查显示：

基于软件重用的软件产品线的发展使开发时间缩短了　　1.5 — 2 倍，维护成本降低了　2-5　倍，

软件质量提高了　5— 10 倍，开发成本降低了　12% — 15%

4、软件体系结构的生命周期模型和软件生命周期模型有什么关系　　？（补充）

软件生命周期　同任何事物一样　,一个软件产品或软件系统也要经历孕育、　　诞生、成长、成熟、

衰亡等阶段，一般称为软件生存周期（软件生命周期）　　。

软件生命周期　(SDLC ，软件生存周期　)是软件的产生直到报废的生命周期，周期内有问题定

义、可行性分析、总体描述、系统设计、编码、调试和测试、验收与运行、维护升级到废弃

等阶段，这种按时间分程的思想方法是软件工程中的一种思想原则，　　即按部就班、 逐步推进，

每个阶段都要有定义、工作、审查、形成文档以供交流或备查，以提高软件的质量。但随着

新的面向对象的设计方法和技术的成熟，软件生命周期设计方法的指导意义正在逐步减

少。 软件生命周期模型　通俗说，就是软件开发过程中所遵循的模式。它运用了结构化的设

计思想； 具有严格的顺序性和依赖性，　必须等待前一阶段的工作完成之后，　才能开始后一阶

段的工作，并且其一阶段的输出就是后一阶段的输入。具体有：　　瀑布（ waterfall　）模型、

原型（ prototyping　）模型、增量（ incremental　）模型、螺旋（ spiral ）模型、快速应用开

发（ RAD ）模型、渐进式模型等。

软件体系结构生命周期模型是对软件生命周期模型的重用，　　是软件生命周期模型的某一阶段

的细化。 它也继承了结构化设计思想，　运用了面向过程的设计方法。　它也具有顺序性和依赖

性。不过它还添加了一些跳转，　当某一阶段比较细致或软件的结构比较简单时可以跳过某一

阶段而经如下一阶段。　软件体系结构生命周期模型包括非形式化的描述；　　体系结构的规范描述和分析；体系结构的求精及其验证；软件体系结构的事实；软件体系结构的演化和扩展；软件体系结构的提供、评价和度量；软件体系结构的终结。

5、 CORBA　架构的技术规范

◎ 接口定义语言（　IDL ）

CORBA 利用 IDL 统一地描述服务器对象（向调用者提供服务的对象）的接口。　　IDL 本身也是面向对象的。　它虽然不是编程语言，但它为客户对象（发出服务请求的对象）提供了语言的独立性，因为客户对象只需了解服务器对象的　　IDL 接口，不必知道其编程语言。

IDL 语言是 CORBA 规范中定义的一种中性语言，　　它用来描述对象的接口，　　而不涉及对象的具体实现。

在 CORBA 中定义了 IDL 语言到 C、C++ 、SmallTalk 和 Java 语言的映射。

◎ 接口池（ IR）

CORBA 的接口池包括了分布计算环境中所有可用的服务器对象的接口表示。　　它使动态搜索可用服务器的接口、动态构造请求及参数成为可能。

◎ 动态调用接口（　DII ）

CORBA 的动态调用接口提供了一些标准函数以供客户对象动态创建请求、　　动态构造请求参数。 客户对象将动态调用接口与接口池配合使用可实现服务器对象接口的动态搜索、　　请求及参数的动态构造与动态发送。当然，只要客户对象在编译之前能够确定服务器对象的 IDL 接口，　CORBA 也允许客户对象使用静态调用机制。显然，静态机制的灵活性虽不及动态机制，但执行效率却胜过动态机制。

◎ 对象适配器（　OA ）

在 CORBA 中，对象适配器用于屏蔽　ORB 内核的实现细节，　为服务器对象的实现者提供抽

象接口，以便他们使用 ORB 内部的某些功能。这些功能包括服务器对象的登录与激活、客户请求的认证等。

6、 C2 概述

◎ C2 和其提供的设计环境 （ Argo ）支持采用基于时间的风格来描述用户界面系统， 并支持使用可替换、可重用的构件开发 GUI 的体系结构。

◎ 在 C2 中，连接件负责构件之间消息的传递，而构件维持状态、执行操作并通过两个名字分别为“ top ”和“ bottom ”的端口和其它的构件交换信息。

◎ 每个接口包含一种可发送的消息和一组可接收的消息。构件之间的消息要么是请求其它构件执行某个操作的请求消息， 要么是通知其他构件自身执行了某个操作或状态发生改变的通知消息。

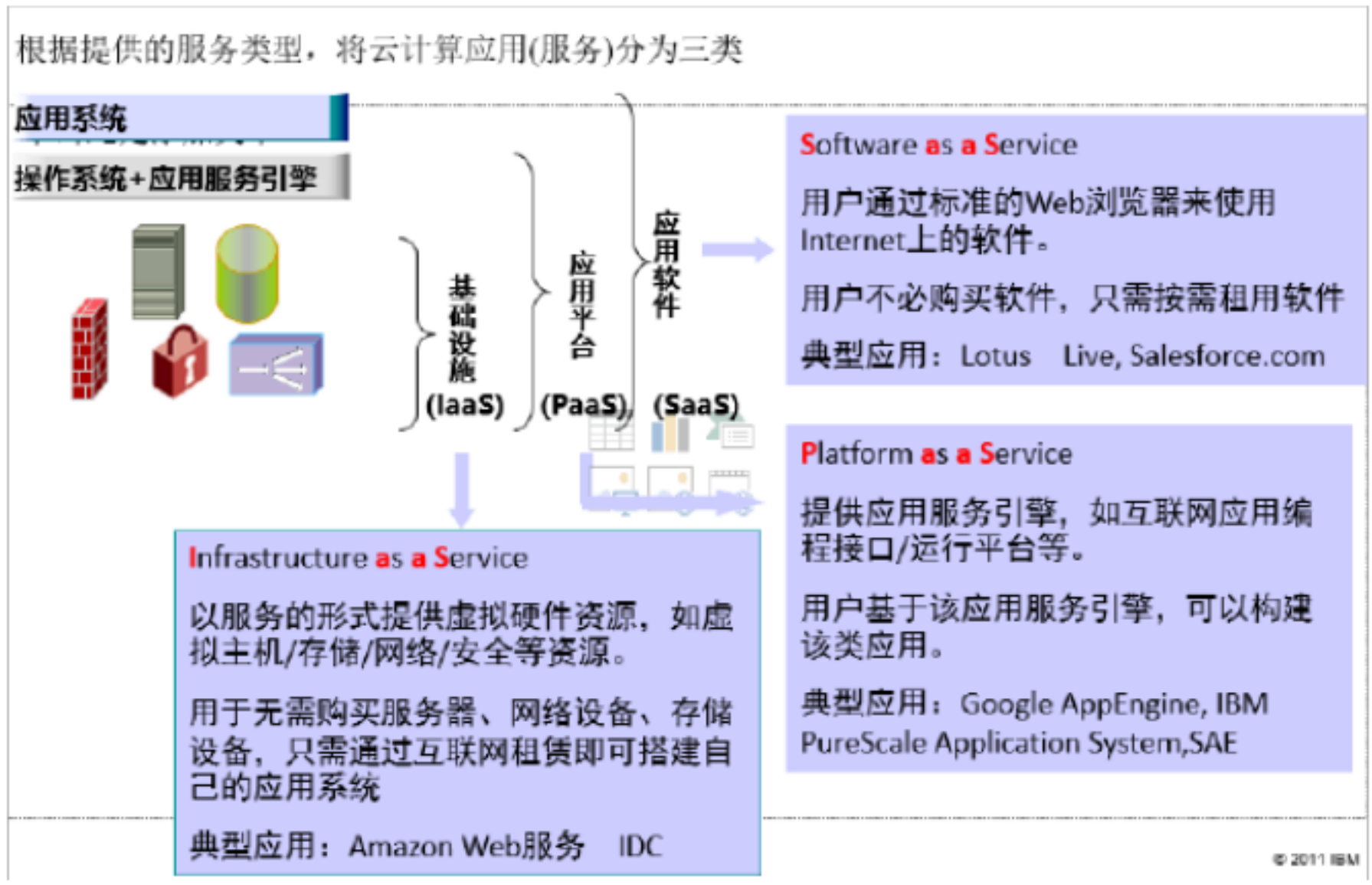
◎ 构件之间的消息交换不能直接进行，而只能通过连接件来完成。每个构件接口最多只能和一个连接件相连，而连接件可以和任意数目的构件或连接件相连。

◎ 请求消息只能向上层传送而通知消息只能向下层传送。

◎ 通知消息的传递只对应于构件内部的操作，而和接收消息的构件的需求无关。

◎ C2 对构件和连接件的实现语言、实现构件的线程控制、构件的部署以及连接件使用的通讯协议等都不加限制。

7、云服务三个层次



计算包括三个层次的服务 :基础架构即服务 （IaaS）, 平台即服务 （PaaS）和软件即服务 （SaaS）。

IaaS 通过互联网提供数据中心、基础架构硬件和软件资源，还可以提供服务器、操作系统、磁盘存储、数据库和 / 或信息资源。

Paas 则提供了基础架构，软件开发者可以在这个基础架构之上建设新的应用，或者扩展已有的应用，同时却不必购买开发、质量控制或生产服务器。

SaaS 是最为成熟、最出名，也是得到最广泛应用的一种云计算。它是一种软件分布模式，在这种模式下，应用软件安装在厂商或者服务供应商那里，用户可以通过某个网络来使用这些软件，通常使用的网络是互联网。

IaaS、PaaS 和 SaaS 之间的关系可从两个角度来看：从用户体验角度而言，它们之间的关系是独立的，因为它们面对不同类型的用户；而从技术角度而言，它们并不是简单的继承关系 (SaaS 基于 PaaS，而 PaaS 基于 IaaS)，因为首先 SaaS 可以是基于 PaaS 或者直接部署于 IaaS 之上，其次 PaaS 可以构建于 IaaS 之上，也可以直接构建在物理

资源之上。

IaaS、 PaaS 和 SaaS 这三种模式都采用了外包的方式，以减轻企业负担，降低管理、维护服务器硬件、 网络硬件、 基础架构软件和应用软件的人力成本。 从更高的层次上看，它们都试图去解决同一个商业问题——用尽可能少甚至是为零的资本支出，获得功能、扩展能力、服务和商业价值。

在云计算实施的前期或者在很多场景的时候； 主要关注点在于应用的可靠运行、 快速开发和部署、 机器资源的充分利用、 以及方便的运维等问题； 对于这个时候我们应该定位于主要采用 Iaas 云计算架构（即很依赖于硬件虚拟化技术）和部分采用 Pass 云计算架构来解决 .

1.重点采用 Iaas 云计算架构中的硬件虚拟化技术等技术（服务器虚拟化、网络虚拟化、存储虚拟化）以提高硬件的利用率、降低机房占用空间和功耗。

2.快速和方便地给应用提供应用所需要的服务器资源（ VM) 、网络资源、存储资源。

3.快速和方便地给应用提供应用所需要依赖的平台软件资源, 例如数据库系统 （DB2) 、J2EE 应用服务器（ WAS) 、WEB 服务器（ IHS) 等。

4.快速和方便地自动地把应用部署到相应的硬件环境中。

5. 硬件虚拟化技术（例如 VMWare ， PowerVM,xen 等）的能力需要在这体现。

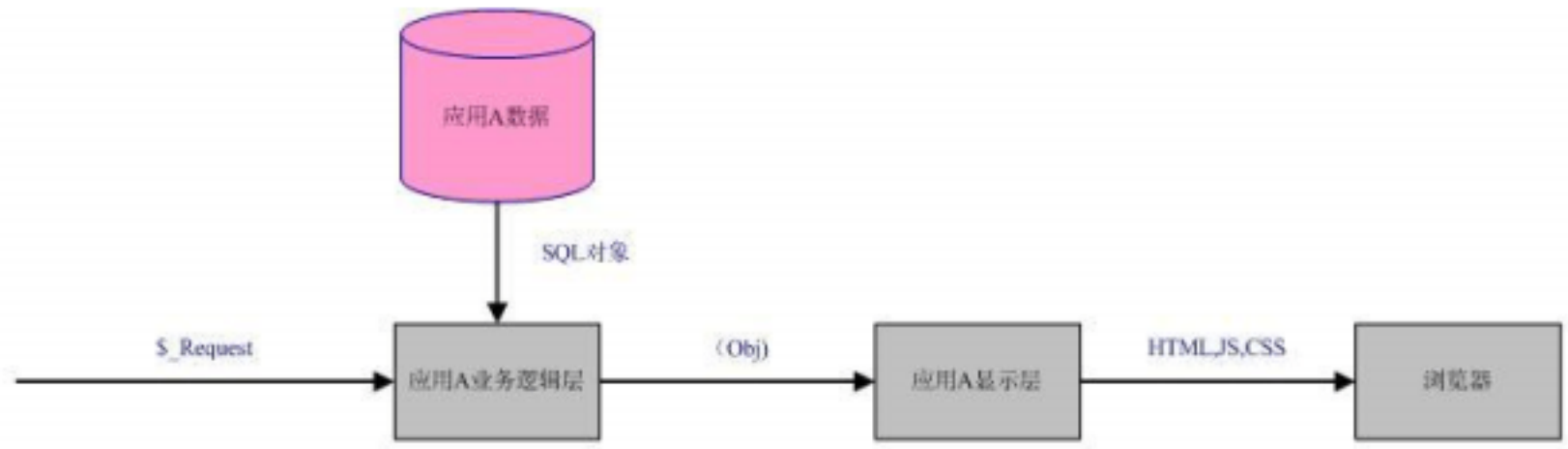
注意：

1.对于在这种情况下，每台服务器所具有的 CPU core 数目和内存数量越大越好。不要弄一些性能较差的机器干这种事情。

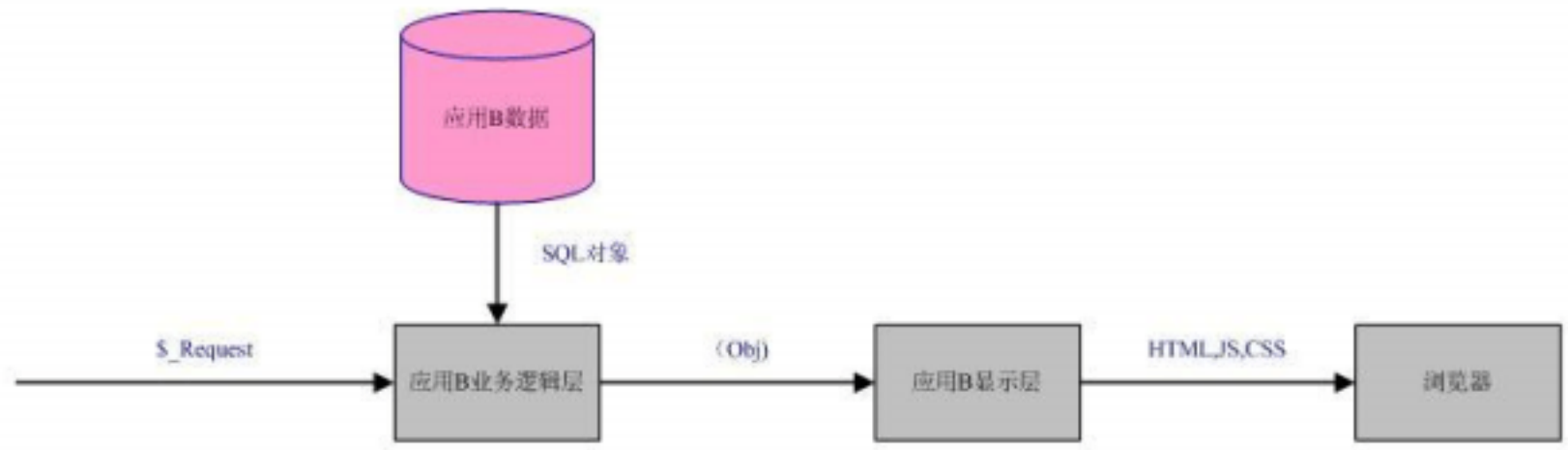
2. SSD 和大内存对数据库性能的提升是很明显的。

3. 这个就是我们通常所说的以大变小

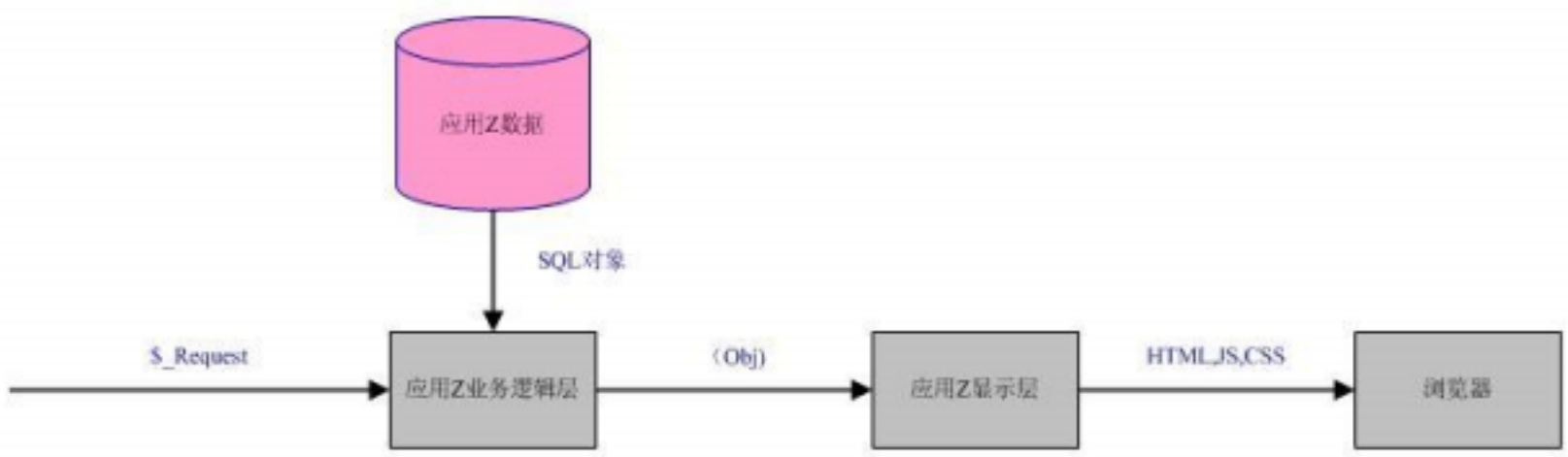
6、现有 IT 系统的主要问题



应用A基础设施



应用B基础设施



应用Z基础设施

这种方式即是以应用为划分的“ 烟囱 ”结构，数据基于应用，并被锁定在应用系统中，形成

一个个数据的孤岛。带来以下一些最基本的问题：

1、数据并没有被作为一个单独的 IT 组成部分被规划和设计，而是作为应用系统的一部分，

由于应用系统的供应商不同， 并且其设计工作也缺乏相互之间的协调， 因此， 数据模型基本

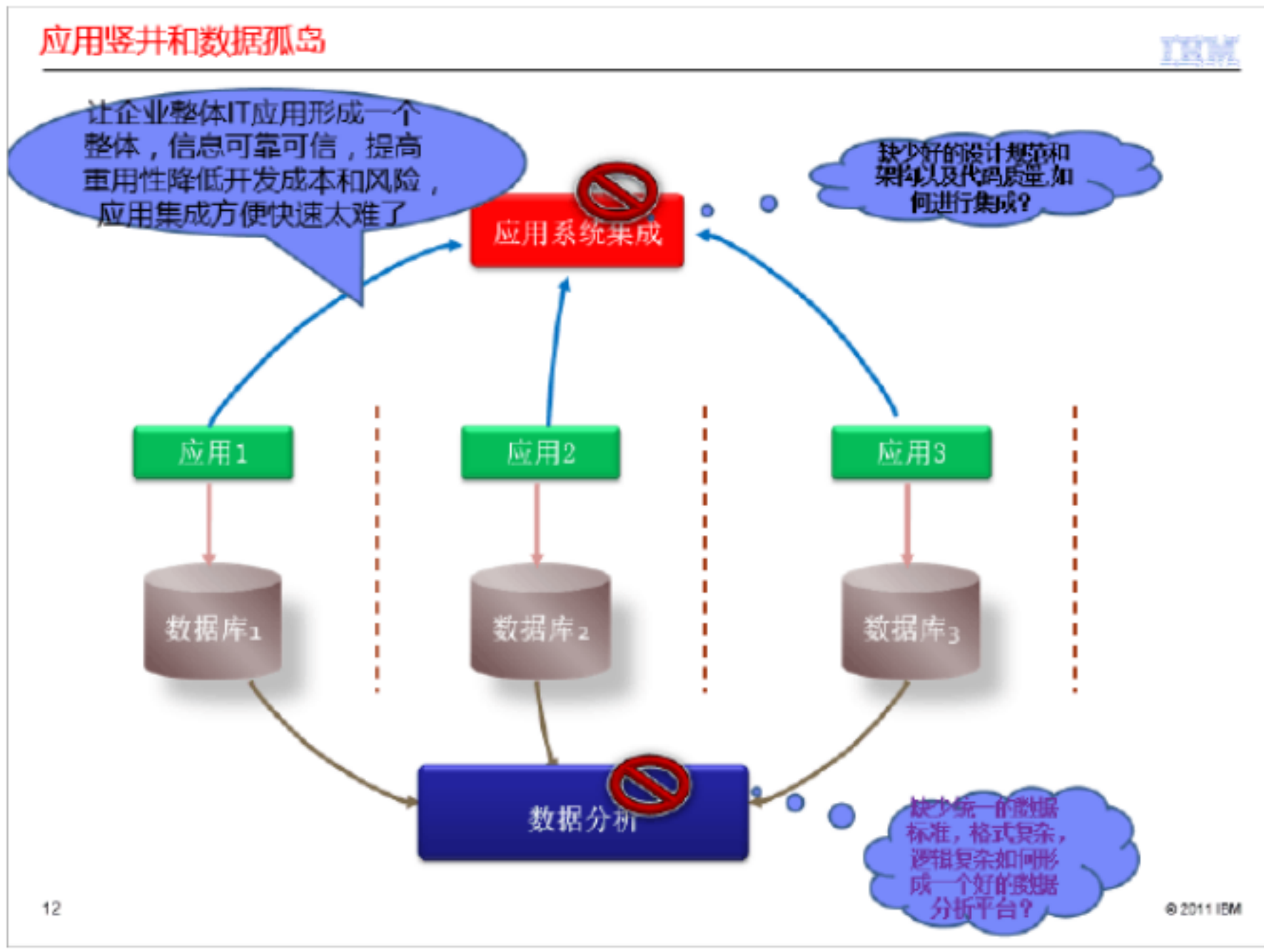
按照各个应用系统的功能需求进行设计和实现。

2、由于缺乏有效的数据共享，在有些业务环节上，一个应用所需的数据无法从相关的其他

应用系统中获得，而只好重复录入。

3、另一方面，由于同一个数据可能存在多个数据源（从多个应用系统中被重复录入），由

此导致了信息的不一致。



这些应用系统按照传统的方式进行设计和开发，从而形成一个个的数据孤岛、应用的竖井等。

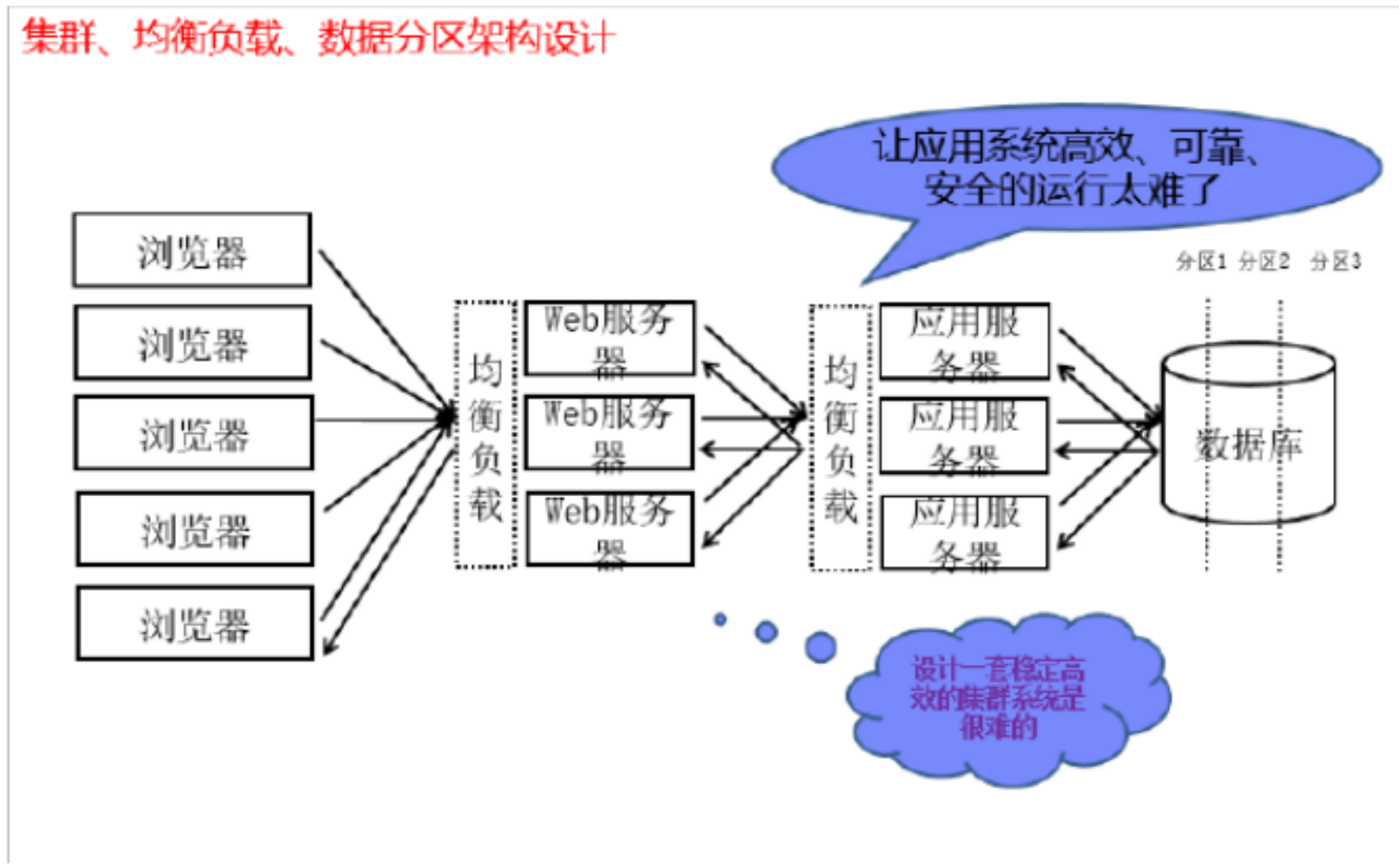
以至不能形成完整的数据视图（例如相互有关联的数据分散于不同的应用系统，数据编码标

准不一致，数据不一致和不可信等；让进行数据的整合和数据分析基本上不太可能）和应用

系统集成复杂度和难度很高（没有好的设计规范和架构以及代码质量，基本上代码重用程度

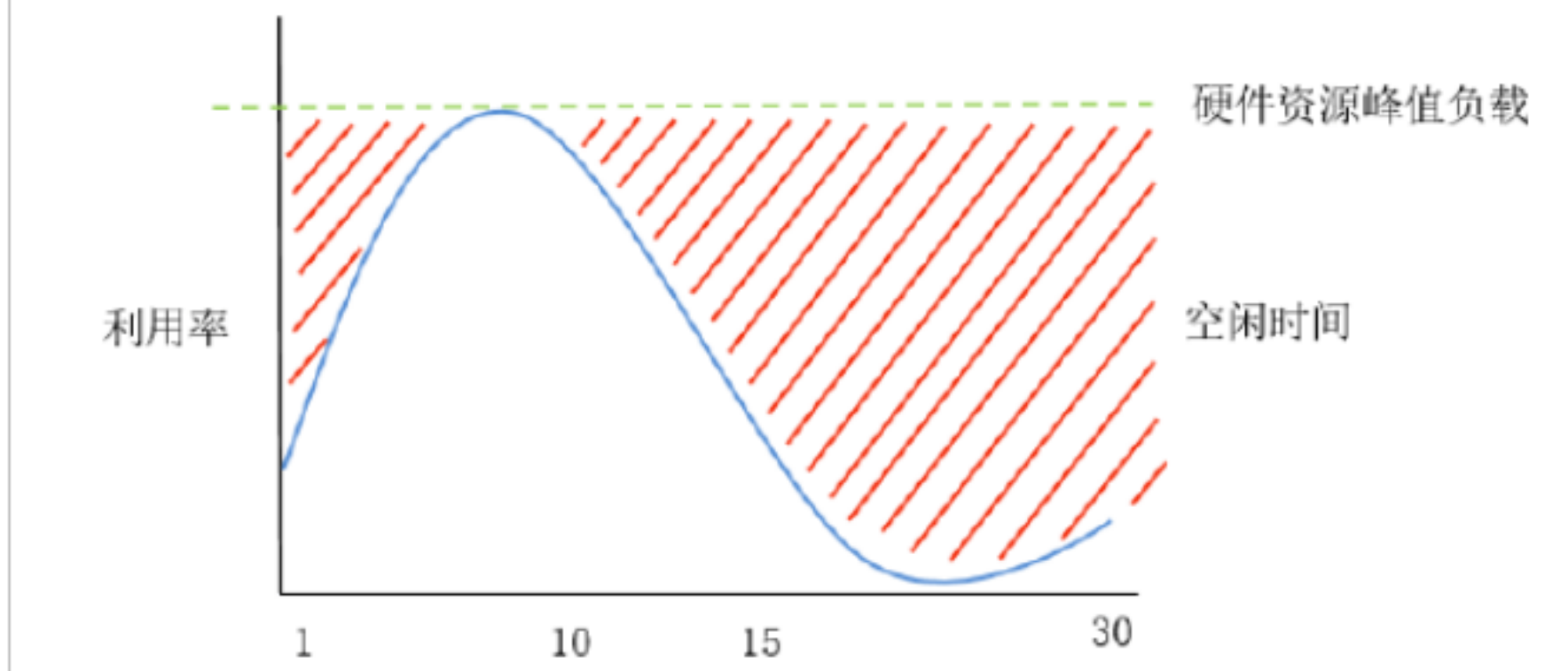
都很低；从界面集成、数据集成、应用集成这三个方向来做都很困难）；甚至出现无法集成

或者为了集成而集成纯粹完成任务而已）



对于一个性能要求很高的系统 （例如高负载量和高数据量的系统, 和将来会做的全省大集中系统和新一代的应用系统） 我们会利用大量的硬件资源和相应的集群技术等技术进行相应的数据分区、集群和均衡负载来应对峰值的访问情况。即利用多台机器和技术形成 Web 服务器的集群, 利用多台机器和技术形成应用服务器的集群, 利用多台机器和技术形成数据库服务器的分区和集群等。 要设计和很好地实施如此复杂的集群牵涉到非常多的技术, 所以我会经常看到当系统压力上来的时候会遇到各种各样的问题并且这方面的问题很严重 （因为很多应用开发商提出的思路基本上是购买更好的硬件来解决又带来了更多的硬件资源的浪费, 而我们都知道仅仅是通过购买好的硬件是解决不了这个问题的） 。

硬件利用率太低，费电、占空间、运维成本高；简而言之太不低碳了



这些应用系统按照传统的项目建设方式，　　即每建设一套业务应用系统基本上都要购买新的硬件设备（例如服务器、存储等）和平台系统软件（例如数据库、中间件等）　　。带来了大量的硬件资源的浪费　（例如大量的服务器利用率低下、　　存储利用率不高和管理复杂）　　和占用大量的空间、电力的浪费、运维成本的提高　　.对于一些高负载和高数据量的应用系统，我们对硬件资源的要求是按照此应用系统高峰值的需要来进行购买以应用此应用的需要，　　但是此应用的高峰期是具有周期性的

7、采用云计算技术后新系统的架构初探

8、大数据　4V　特征和什么是大数据？

什么是大数据（　Big Data　）？

? Volume:　数据量异常庞大，一般达到　　PB 量级

? Variety:　数据呈异构化，数据来源呈多样性

? Velocity:　数据处理要求时效性

? Value:　单个数据无价值，但大规模数据拥有巨大价值　　；

什么是大数据（ Big Data ）？

数据种类的多样性：文字、语音、图片、视频、信息等

？数据对象的多样性：个人信息、个人数据、商业服务数据、社会公共数据、自然界数据、

物质世界的数据

？数据来源的多样性： 在数据层面打破现实世界的界限， 多家公司的共享替代一家公司的数

据

9、离线批处理模型、内存计算模型、交互计算模型的区别

离线批处理模型： GFS/HDFS/NoSQL/MapReduce ，业界主流模式，技术成熟，数

据规模大，但时效性差

大内存计算模型： Hana,MemCloud ，计算速度快，但需要大规模集中式内存结构支

持（若为分布式则受制于网络传输速度） ，技术成熟度不够

交互式计算模型： Google 有 Dremel, PowerDrill,Apache 有 Drill 通过 data

locality/in-memory buffer/columnar data structure 等技术来提高计算速度，以现

有计算架构和软件技术为基础，具可行性；但目前技术分散，缺乏一个集成平台

10 、大数据总结

互联网大数据已成为云计算、物联网之后的又一新技术热点，已上升到国家战略高度，

具有巨大市场价值；

企业对大数据的需求重点在实时在线计算能力，以支持智能商务应用。现有的迭代批处

理 MapReduce 算法时效性差，难以满足海量数据的实时在线分析；

互联网大数据计算目前尚未有业界认可的主流产品， 技术成熟度不到 30% ， 交互式计算

和大内存计算是值得关注的技术。

11、Hadoop　原理：　HDFS　及　MapReduce

Hadoop　最主要的就是　Core，它又分为　HDFS 和 MapReduce　两个部分，　前者提供分布式数据存储，后者提供任务的分发和归拢。其他组件都是围绕着这两个核心进行工作。

HDFS 的设计初衷：

一次写入多次读取不支持文件并发写入不支持文件修改　　.

HDFS 的适用范围：

存储并管理　PB 级数据处理非结构化数据注重数据处理的吞吐量，　　对延时不敏感应用模式为一次写入多次读取存取模式　.

12、设计 SOA 架构图

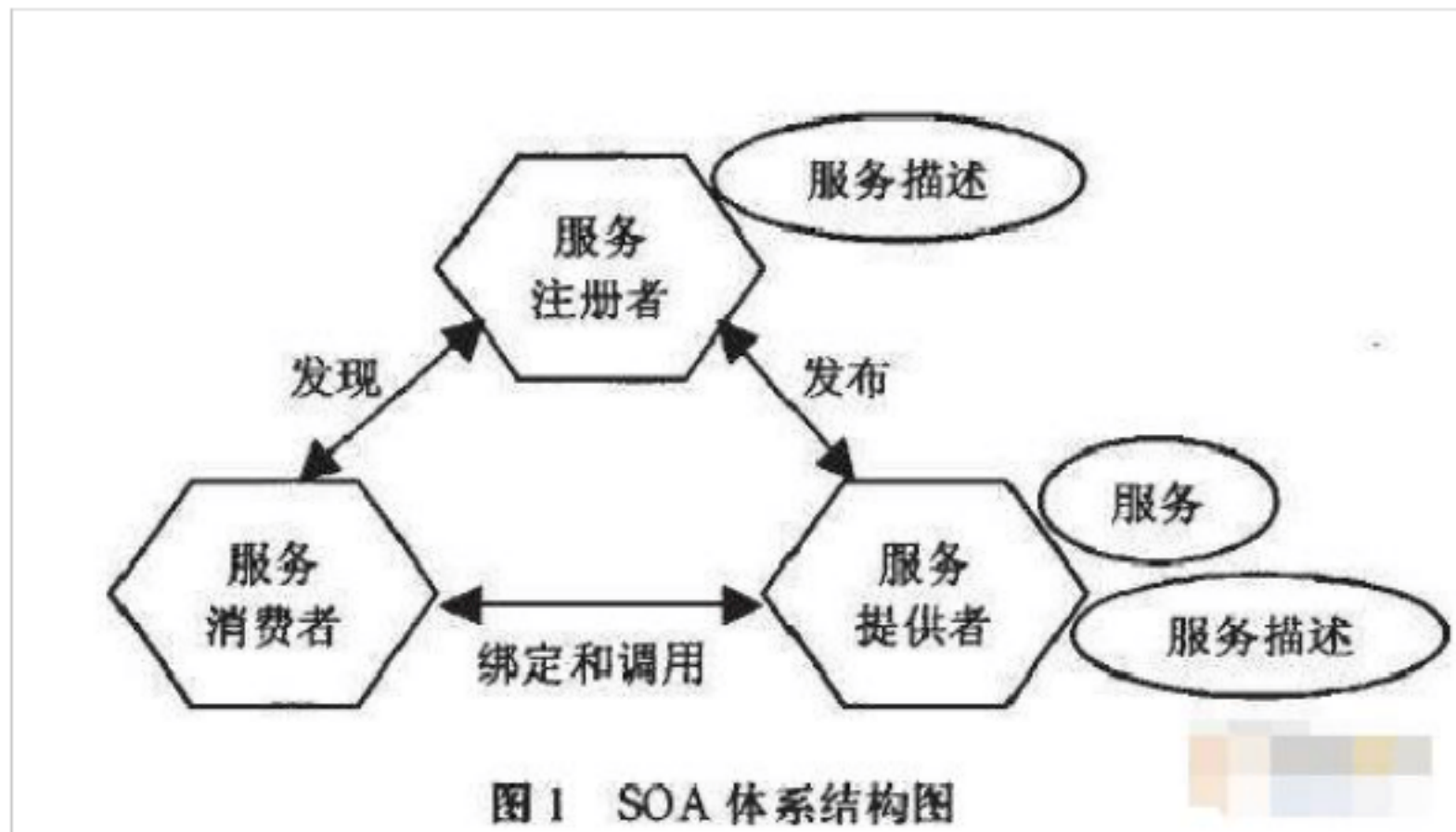
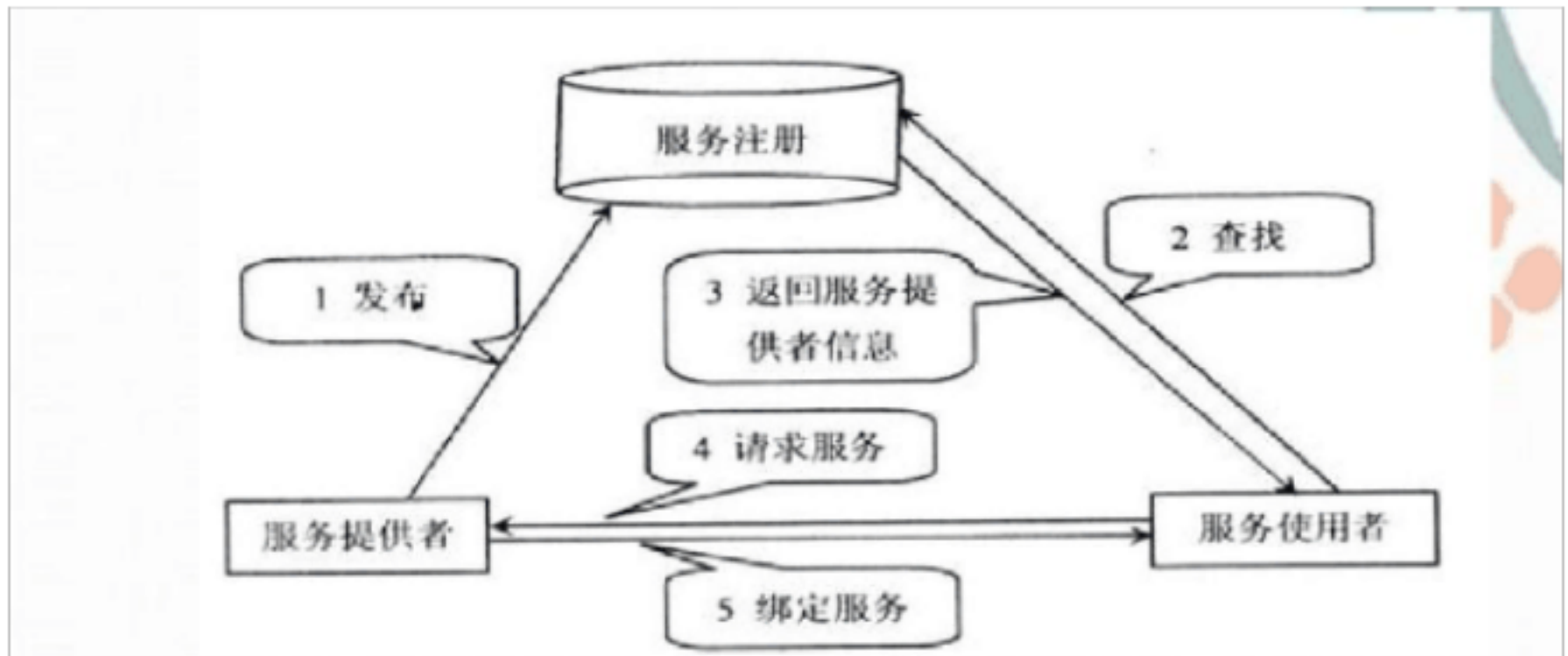
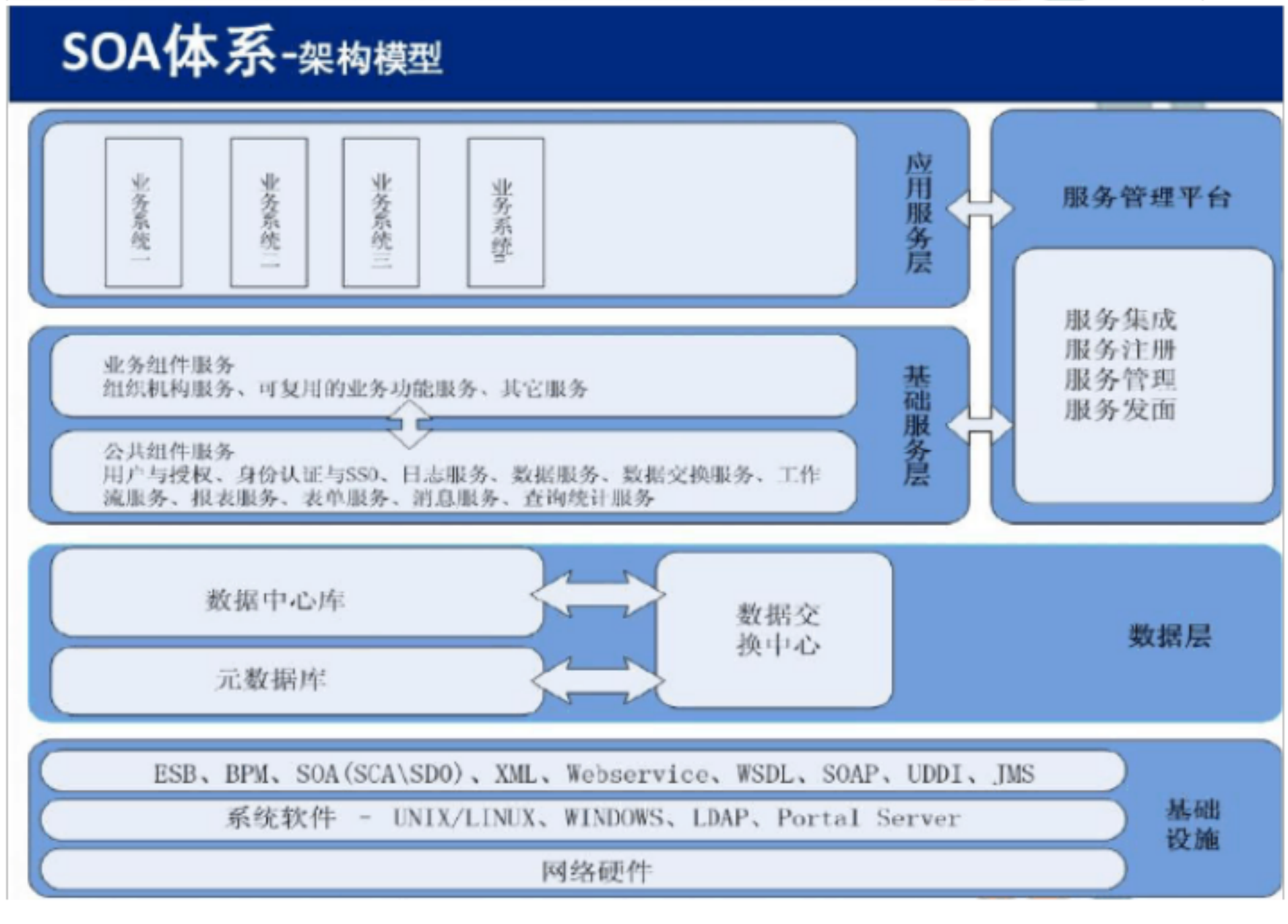


图 1　SOA 体系结构图

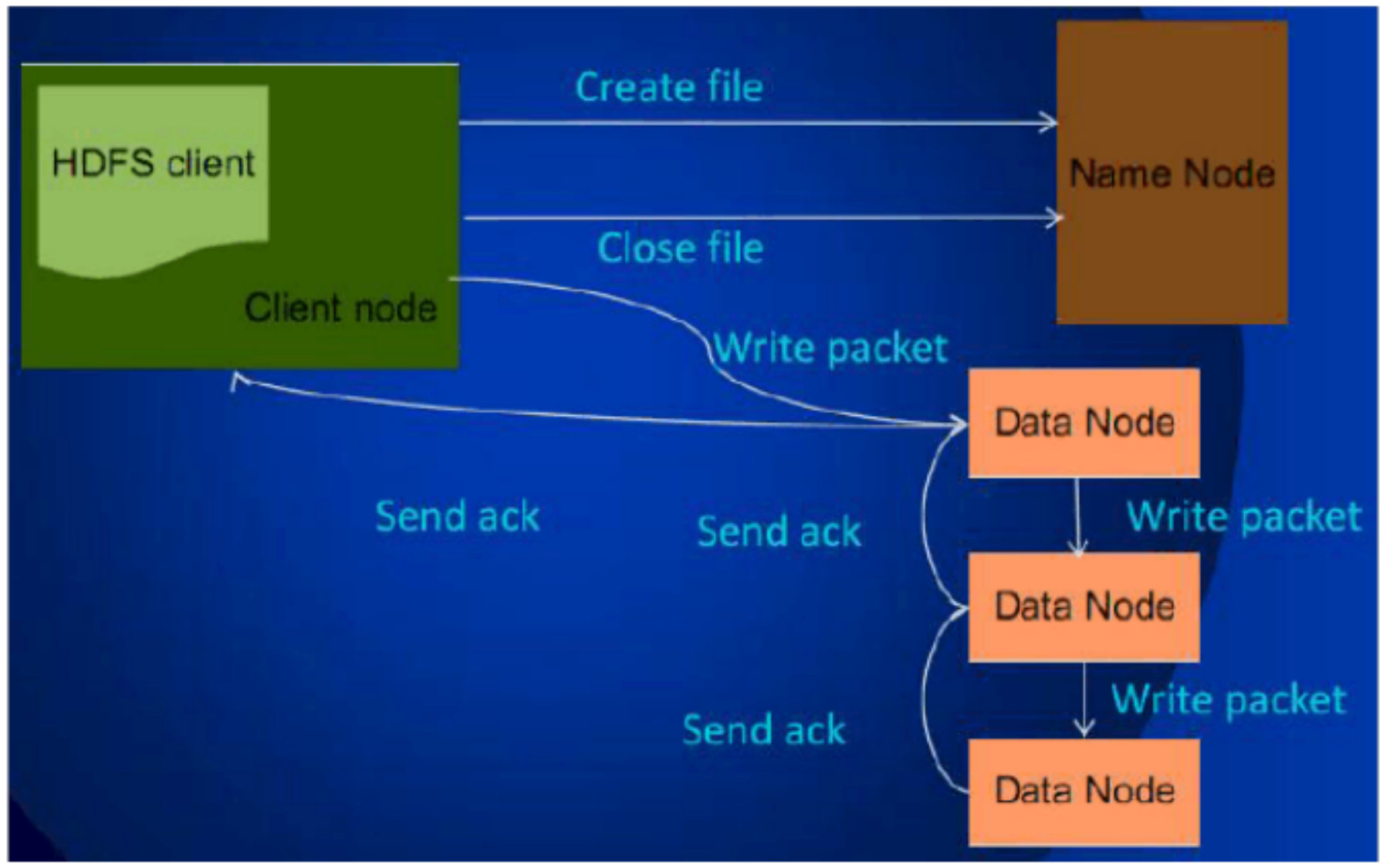


(1)服务提供者：服务提供者是一个可通过网络寻址的实体，它接受和执行来自使用者的请求；
(2)服务使用者：服务使用者是一组使用服务提供者所提供的一项或多项服务的组件；
(3)服务储备库：服务储备库包含服务的描述，服务提供者在该储备库中注册其服务，而服务使用者访问该储备库已发现的所提供的服务。

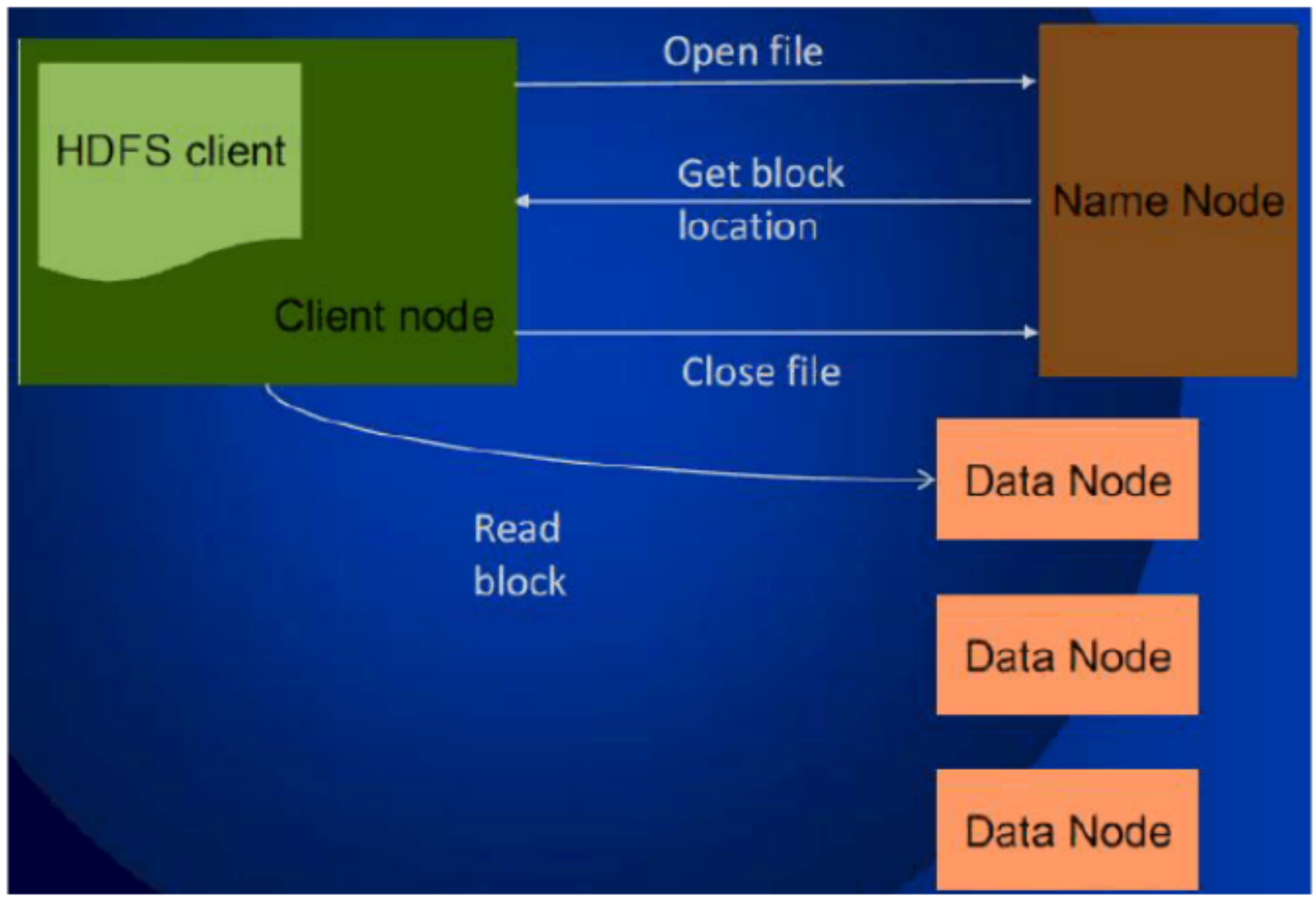


13、HDFS 处理过程

（1）HDFS 如何写文件：



（2）HDFS 如何读文件：



（3）HDFS 上传文件流程：

当客户端上传数据时，先向 NamaNode 请求创建文件，然后 NameNode 会返回存储的 DataNode 节点和位置信息。随后客户端先通过管道将文件传到本地硬盘，每凑满指定大

小（默认是 64M ），再一起上传到一个 DataNode ， DataNode 收到文件后会返回确认信息，并以 4K 为单位传到下一 DataNode 。该文件上传方式被称为“ 流水式 ” 。

14 、 MapReduce 处理过程

MapReduce分布式处理技术——Map端

map: (k1, v1) → list(k2, v2)

- 输入：键值对(k1, v1)表示的数据
- 处理：文档数据记录（如文本文件中的行，或数据表格中的行）将以“键值对”形式传入map函数；map函数将处理这些键值对，并以另一种键值对形式输出处理的一组键值对中间结果list(k2, v2)。
- 输出：键值对(k2, v2)表示的一组中间数据
- 备注: list(k2, v2) 表示有一个或多个键值对组成的列表

MapReduce分布式处理技术——Reduce端

reduce: (k2, list(v2)) → list(k3, v3)

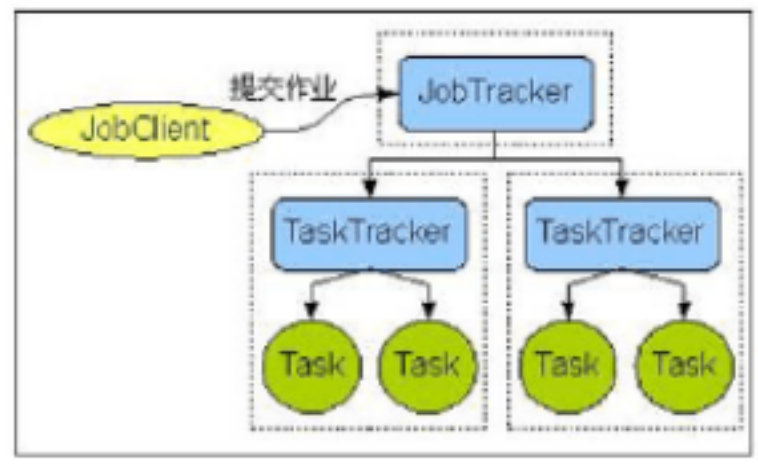
- 输入：由map输出的一组键值对list(k2, v2) 将被进行合并处理，同样主键下的不同数值合并会到一个list(v2)中，故reduce的输入为(k2, list(v2))。
- 处理：对传入的中间结果列表数据进行某种整理或进一步的处理，并产生最终的某种形式的结果输出list(k3, v3)。
- 输出：最终输出结果list(k3, v3)。

MapReduce分布式处理技术

函数	输入	输出
Map	(k1, v1)	List(k2, v2)
Reduce	(k2, List(v2))	List(k3, v3)

- 各个map函数对所划分的数据并行处理，从不同的输入数据产生不同的中间结果输出；
- 各个reduce各自并行计算，各自负责处理不同的中间结果数据集合；
- 进行reduce处理之前，须等到所有的map函数做完，并且在进入reduce前会对map的中间结果数据进行整理(Shuffle)，保证将map的结果发送给对应的reduce；
- 最终汇总所有reduce的输出结果即可获得最终结果

MapReduce分布式处理技术——框架



- MapReduce框架是由一个单独运行在主节点上的JobTracker 和运行在每个集群从节点上的TaskTracker共同组成的。
- 主节点负责调度构成一个作业的所有任务，这些任务分布在不同的从节点上。主节点监控它们的执行情况，并且重新执行之前失败的任务。

15 、 MapReduce 分布式处理技术 -实现机制

(1) MapReduce 函数库首先把输入文件分成 M 块，每块大概 16MB 到 64MB 。接着在集群的机器上执行处理程序。

MapReduce 算法运行过程中有一个主控程序，称为 master 。主控程序会产生很多作业程序，称为 worker 。并且把 M 个 map 任务和 R 个 reduce 任务分配给这些 worker ，让它们去完成。

(2) 被分配了 map 任务的 worker 读取并处理相关的输入 (这里的输入是指已经被切割的输入小块 splite) 。它处理输入的数据, 并且将分析出的键 / 值 (key/value) 对传递给用户定义的 reduce() 函数。 map() 函数产生的中间结果键 / 值 (key/value) 对暂时缓冲到内存。

(3) map() 函数缓冲到内存的中间结果将被定时刷写到本地硬盘，这些数据通过分区函数分成 R 个区。这些中间结果在本地硬盘的位置信息将被发送回 master ，然后这个 master 负责把这些位置信息传送给 reduce() 函数的 worker 。

(4) 当 master 通知了 reduce() 函数的 worker 关于中间键 / 值 (key/value) 对的位置时, worker 调用远程方法从 map() 函数的 worker 机器的本地硬盘上读取缓冲的中间数据。当 reduce() 函数的 worker 读取到了所有的中间数据, 它就使用这些中间数据的键 (key) 进行排序，这样可以使得相同键 (key) 的值都在一起。

(5) reduce() 函数的 worker 根据每一个中间结果的键 (key) 来遍历排序后的数据，并且把键 (key) 和相关的中间结果值 (value) 集合传递给 reduce() 函数。 reduce() 函数的 worker 最终把输出结果存放在 master 机器的一个输出文件中。

(6) 当所有的 map 任务和 reduce 任务都已经完成后, master 激活用户程序。在这时, MapReduce 返回用户程序的调用点。

(7) 当以上步骤成功结束以后, MapReduce 的执行数据存放在总计 R 个输出文件中 (每个输出文件都是由 reduce 任务产生的，这些文件名是用户指定的)。通常，用户不需要将这 R 个输出文件合并到一个文件，他们通常把这些文件作为输入传递给另一个 MapReduce 调用，或者用另一个分布式应用来处理这些文件, 并且这些分布式应用把这些文件看成为输入文件由于分区 (partition) 成为的多个块文件。

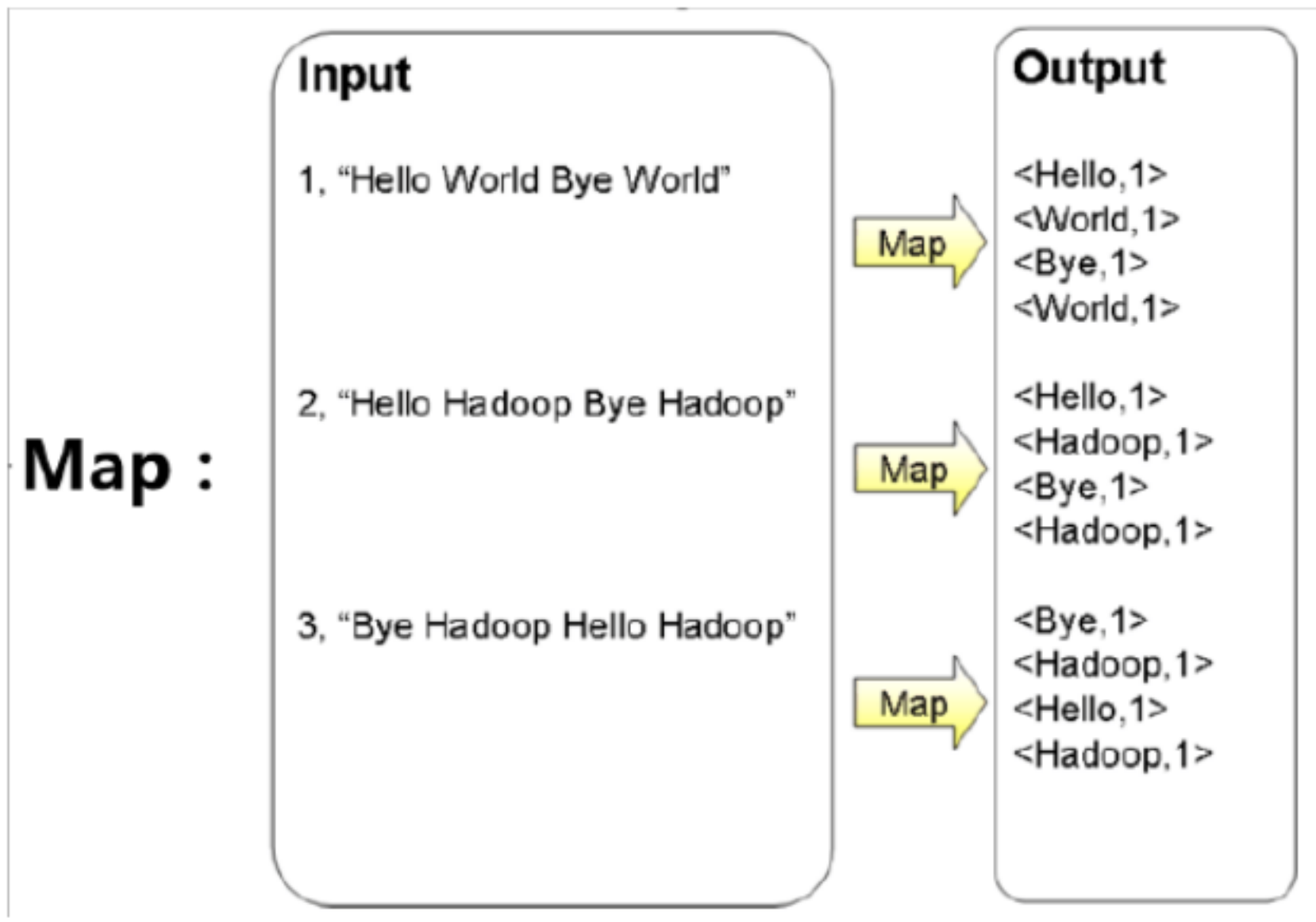
16 、 MapReduce 分布式处理技术 -实例 - 单词统计 WordCount

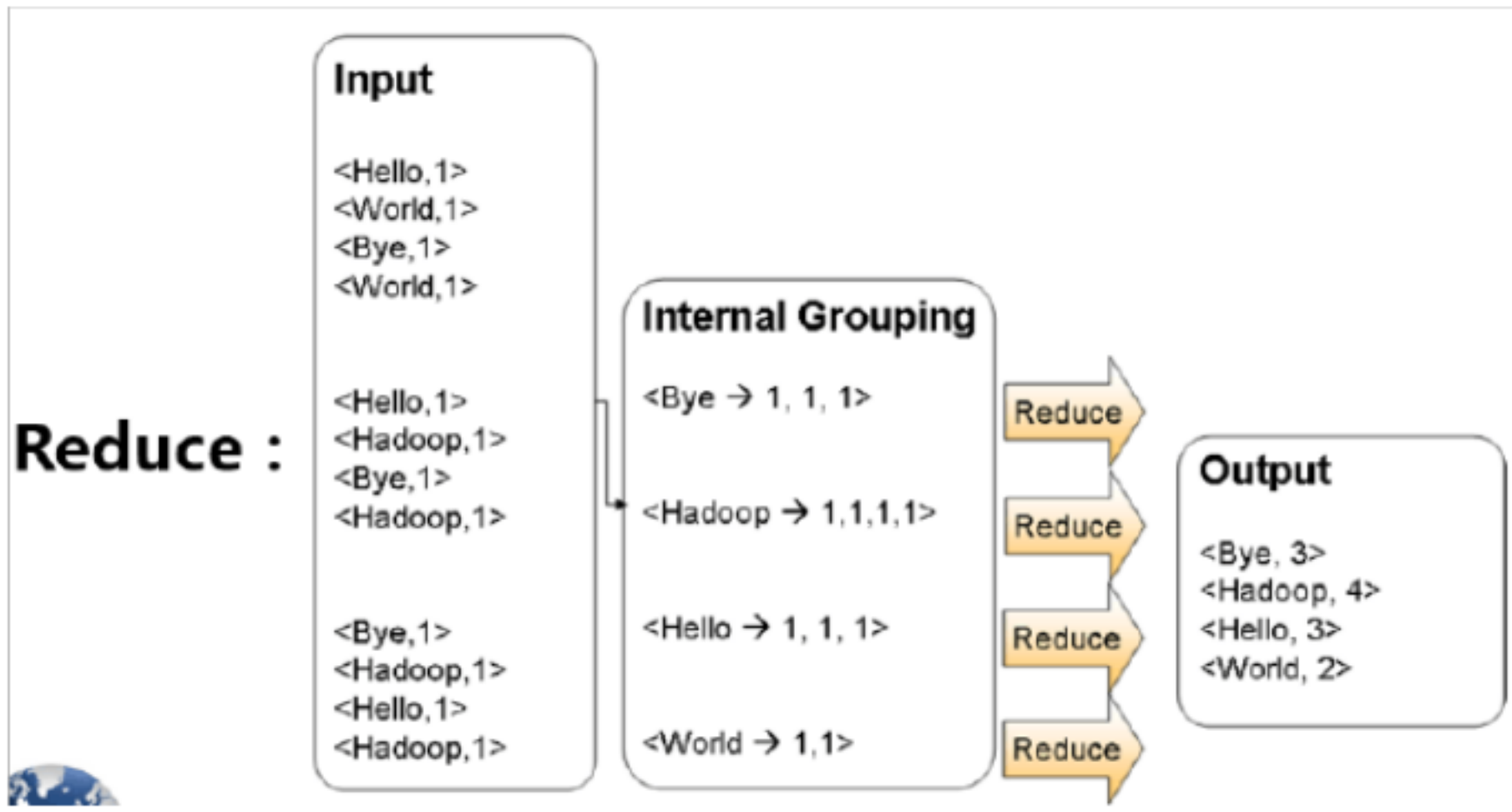
问题描述：统计文本中所出现单词的次数。

解决思路：需符合 Map 、Reduce 各自的输入、输出格。

Map 端：输入为 (k1, v1), 以文本行号为 k1，以行内容为 v1；输出为 list(k2, v2) ，每有一个单词就输出一个 (word, 1) 。

Reduce 端：输入为 (k2, list(v2)) ，将 list(v2) 中所有数字相加即可得到单词次数；输出为 list(k3, v3) ，即最终的结果： (单词，单词次数)。





以下是书上的案例：

示例：WordCount

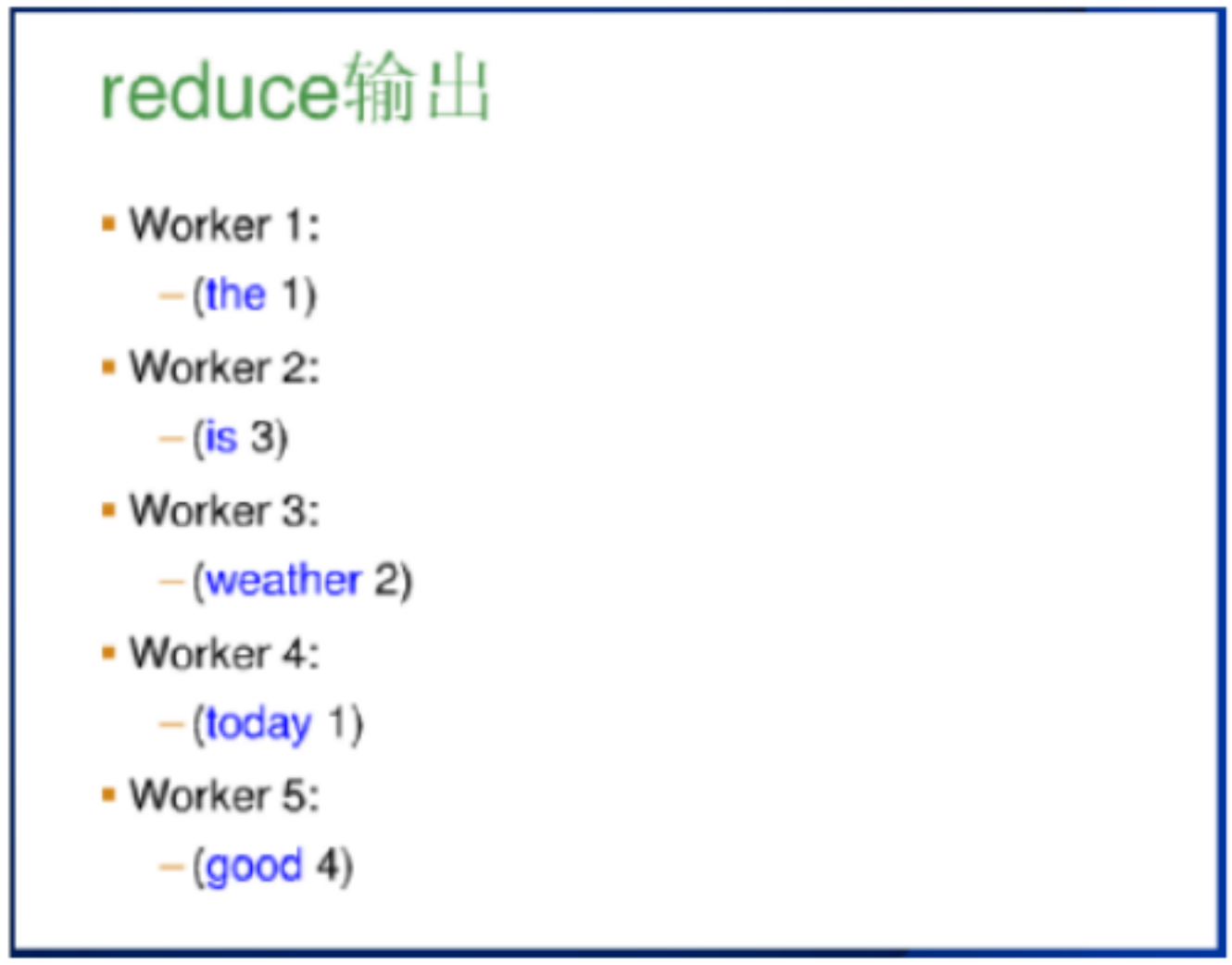
- 源数据
  - Page 1:
    - the weather is good
  - Page 2:
    - today is good
  - Page 3:
    - good weather is good

map 输出

- Worker 1:
  - (the 1), (weather 1), (is 1), (good 1).
- Worker 2:
  - (today 1), (is 1), (good 1).
- Worker 3:
  - (good 1), (weather 1), (is 1), (good 1).

reduce 的输入

- Worker 1:
  - (the 1)
- Worker 2:
  - (is 1), (is 1), (is 1)
- Worker 3:
  - (weather 1), (weather 1)
- Worker 4:
  - (today 1)
- Worker 5:
  - (good 1), (good 1), (good 1), (good 1)



17 、 MapReduce 分布式处理技术 -实例 - 文档倒排索引算法

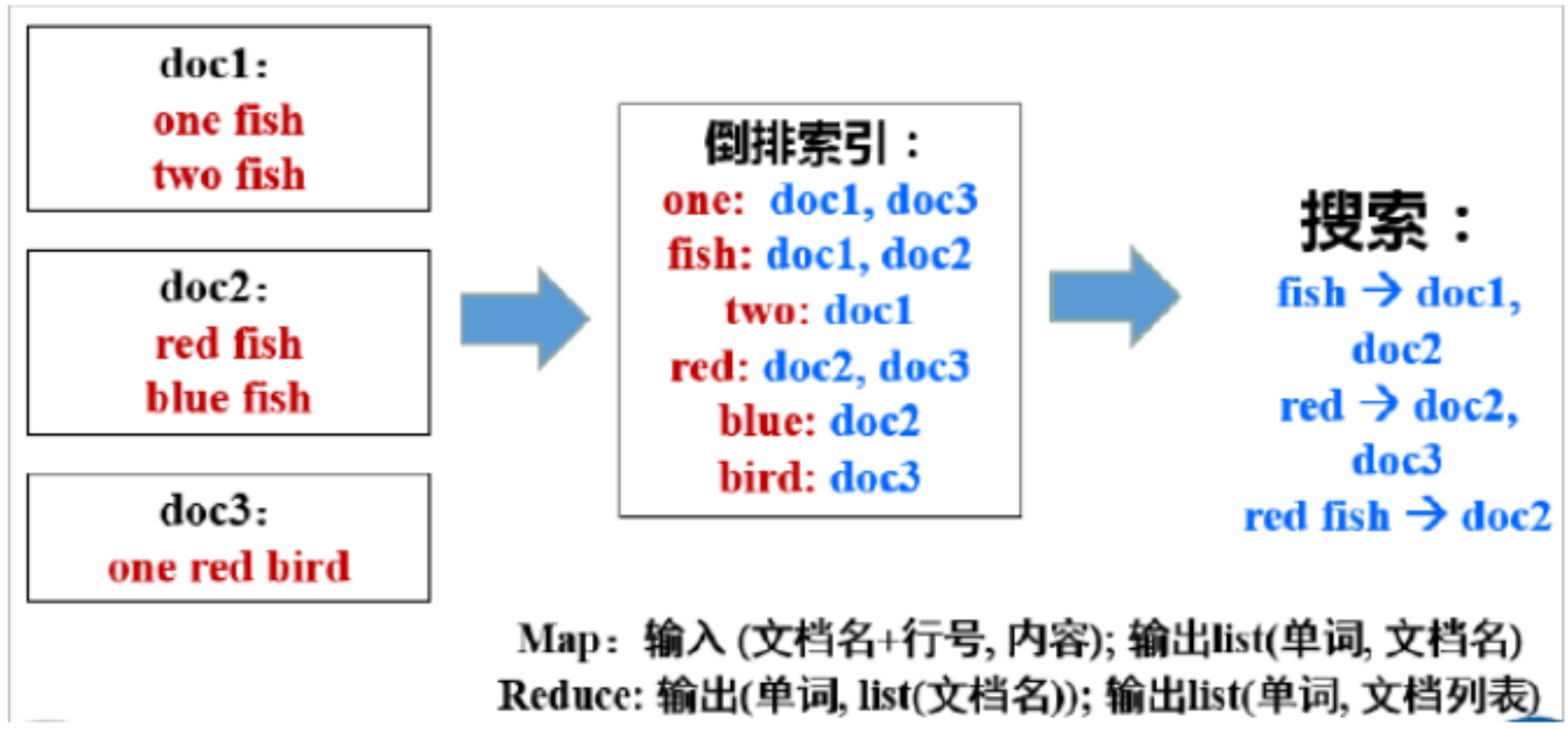
问题描述：给出气象站历年每天的数据，要提取出每年的最高气温。数据示例：

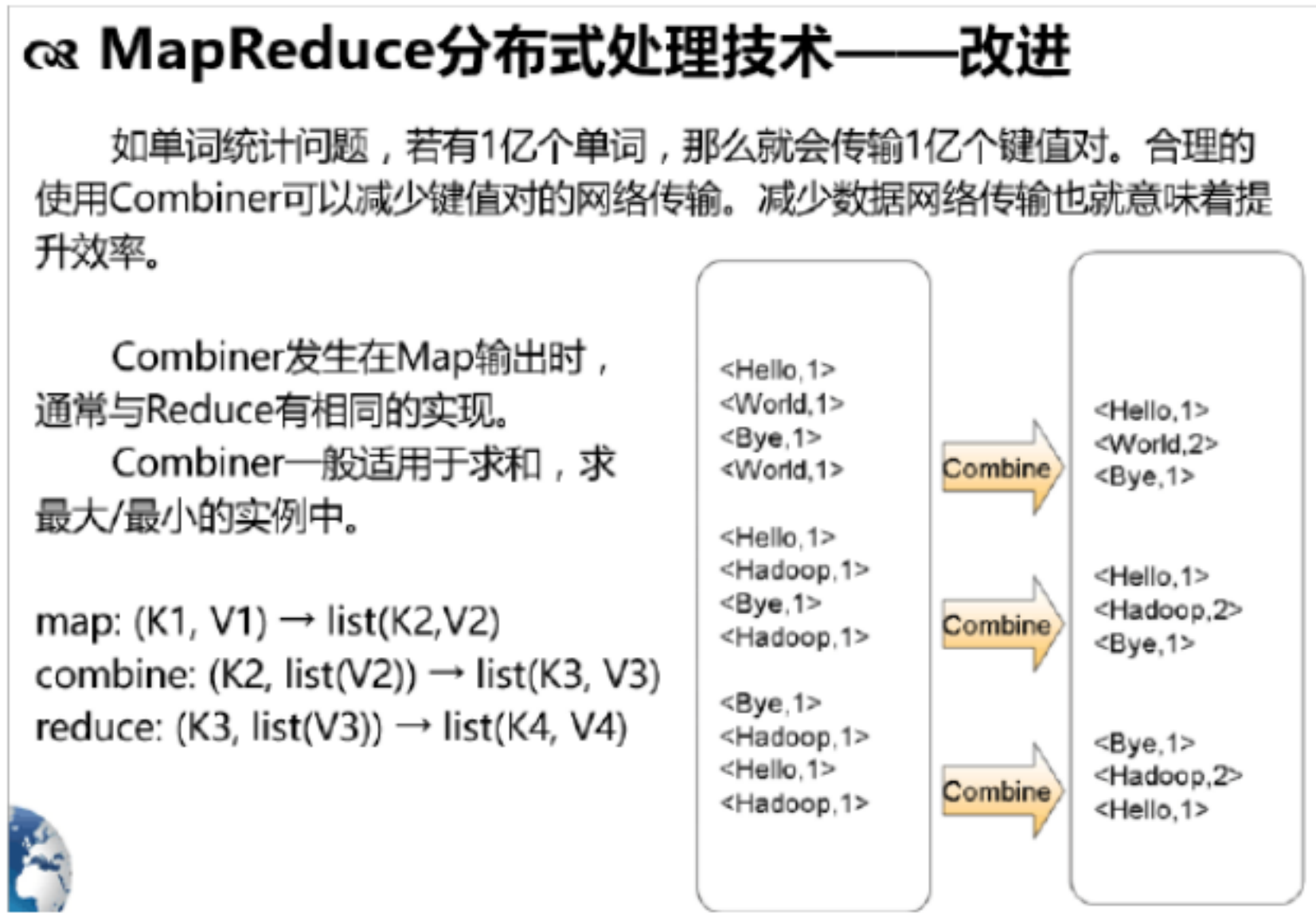
0067011990999991950051507004 …..99999N9+00221+99999 …

解决思路 -Map 端： 输入为 (k1, v1): (行号， 内容)；输出为 list(k2, v2): (年份， 温度)，例如 (1950, 22) 。

解决思路 -Reduce 端： 输入为 (k2, list(v2): 如 (1950, [22, -11]) ，对 list(v2) 排序可得到高气温；输出为 list(k3, v3) ，即最终的结果： (年份，最高气温)。

问题描述： Inverted Index(倒排索引)是目前几乎所有支持全文检索的搜索引擎都要依赖的一个数据结构。基于索引结构，给出一个词 (term) ，能取得含有这个 term 的文档列表 (the list of documents)





18 、区域系统架构扩展方案

19 、中间件的优点

1、缩短应用开发周期； 2、减少项目开发的风险，已经过大量测试完善； 3、应用系统质量的可维护性； 4、增加产品的吸引力； 5、透明与其它应用程序的交互； 6、与运行平台提供网络通信服务无关； 7、具有良好的可靠性和可用性； 8、良好的可扩展性；

20 、架构设计的基本准则

1、抽象； 2、封装； 3、信息隐藏； 4、模块化； 5、注意点分离； 6、内聚与耦合

CORBA

1、如何实现需求端到服务器端两者之间的对接：

答：以查询名录的行为方式去找到服务。

2、中介代理的作用：连接需求端到服务器端的桥梁。

3、 中 介在架构中的作用：

a) 客户和服务器之间无需直接了解；

b) 客户和服务器不必一对一；

c) 客户可以在运行的时候定位服务并交互